


[> home](#) [> about](#) [> feedback](#) [> login](#)

US Patent & Trademark Office



Try the *new* Portal design
Give us your opinion after using it.

Search Results

Search Results for: [edge<AND>((hardware<AND>((hotspot<AND>((profile))))))]

Found 30 of 121,259 searched.

Search within Results

 [> Advanced Search](#)
[> Search Help/Tips](#)Sort by: [Title](#) [Publication](#) [Publication Date](#) [Score](#) [Binder](#)Results 1 - 20 of 30 [short listing](#)
 Prev Page 1 2 Next Page
1 [Overcoming the challenges to feedback-directed optimization \(Keynote Talk\)](#) 83%

Michael D. Smith

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN workshop on Dynamic and adaptive compilation and optimization January 2000
Volume 35 Issue 7

Feedback-directed optimization (FDO) is a general term used to describe any technique that alters a program's execution based on tendencies observed in its present or past runs. This paper reviews the current state of affairs in FDO and discusses the challenges inhibiting further acceptance of these techniques. It also argues that current trends in hardware and software technology have resulted in an execution environment where immutable executables and traditional static optimizations are ...

2 [Compilers II: Inter-procedural stacked register allocation for itanium® like architecture](#) 82%

Liu Yang , Sun Chan , G. R. Gao , Roy Ju , Guei-Yuan Lueh , Zhaoqing Zhang

Proceedings of the 17th annual international conference on Supercomputing June 2003

A hardware managed register stack, Register Stack Engine (RSE), is implemented in Itanium® architecture to provide a unified and flexible register structure to software. The compiler allocates each procedure a register stack frame with its size explicitly specified using an *alloc* instruction. When the total number of registers used by the procedures on the call stack exceeds the number of physical registers, RSE performs automatically register overflows and fills to ensure that the c ...

3 [Online feedback-directed optimization of Java](#) 80%

Matthew Arnold , Michael Hind , Barbara G. Ryder

ACM SIGPLAN Notices , Proceedings of the 17th ACM conference on Object-oriented

programming, systems, languages, and applications November 2002

Volume 37 Issue 11

This paper describes the implementation of an online feedback-directed optimization system. The system is fully automatic; it requires no prior (offline) profiling run. It uses a previously developed low-overhead instrumentation sampling framework to collect control flow graph edge profiles. This profile information is used to drive several traditional optimizations, as well as a novel algorithm for performing feedback-directed control flow graph node splitting. We empirically evaluate this syst ...

4 Creating and preserving locality of java applications at allocation and garbage collection times 80%

Yefim Shuf , Manish Gupta , Hubertus Franke , Andrew Appel , Jaswinder Pal Singh
ACM SIGPLAN Notices , Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications November 2002

Volume 37 Issue 11

The growing gap between processor and memory speeds is motivating the need for optimization strategies that improve data locality. A major challenge is to devise techniques suitable for pointer-intensive applications. This paper presents two techniques aimed at improving the memory behavior of pointer-intensive applications with dynamic memory allocation, such as those written in Java. First, we present an allocation time object placement technique based on the recently introduced notion of p ...

5 Dynamic hot data stream prefetching for general-purpose programs 80%

Trishul M. Chilimbi , Martin Hirzel

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation May 2002

Volume 37 Issue 5

Prefetching data ahead of use has the potential to tolerate the grow-ing processor-memory performance gap by overlapping long latency memory accesses with useful computation. While sophisti-cated prefetching techniques have been automated for limited domains, such as scientific codes that access dense arrays in loop nests, a similar level of success has eluded general-purpose pro-grams, especially pointer-chasing codes written in languages such as C and C++. We address this problem by describing ...

6 Partial method compilation using dynamic profile information 80%

John Whaley

ACM SIGPLAN Notices , Proceedings of the OOPSLA '01 conference on Object Oriented Programming Systems Languages and Applications October 2001

Volume 36 Issue 11

The traditional tradeoff when performing dynamic compilation is that of fast compilation time versus fast code performance. Most dynamic compilation systems for Java perform selective compilation and/or optimization at a method granularity. This is the not the optimal granularity level. However, compiling at a sub-method granularity is thought to be too complicated to be practical. This paper describes a straightforward technique for performing compilation and optimizations at a finer, sub-metho ...

7 A study of exception handling and its dynamic optimization in Java 80%

Takeshi Ogasawara , Hideaki Komatsu , Toshio Nakatani


ACM SIGPLAN Notices , Proceedings of the OOPSLA '01 conference on Object Oriented Programming Systems Languages and Applications October 2001

Volume 36 Issue 11

Optimizing exception handling is critical for programs that frequently throw exceptions. We observed that there are many such exception-intensive programs in various categories of Java programs. There are two commonly used exception handling techniques, stack unwinding optimizes the normal path, while stack cutting optimizes the exception handling path. However, there has been no single exception handling technique to optimize both paths.

8 Visualizing the performance of higher-order programs

80%

 Oscar Waddell , J. Michael Ashley


ACM SIGPLAN Notices , Proceedings of the 1998 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering July 1998

Volume 33 Issue 7

Profiling can provide the information needed to identify performance bottlenecks in a program, but the programmer must understand its relation to the program source in order to use this information. This is difficult due to the tremendous volume of data collected. Moreover, program transformations such as macro expansion and procedure inlining can obscure the relationship between the source and object code. Higher-order programs present additional challenges due to complex control flow and because ...

9 K9: a simulator of distributed-memory parallel processors

80%


 P. Beadle , C. Pommerell , M. Annaratone

Proceedings of the 1989 ACM/IEEE conference on Supercomputing August 1989

K9 is a software package for the simulation and performance evaluation of distributed-memory parallel processors (DMPPs). It is written in C++ and runs on Sequent Symmetry and SUN-3. K9 provides the user with four building-blocks (processor cells, communication channels, multi-port shared-memories, and I/O processors), and one abstraction mechanism (the DMPP interconnection topology). Application code for K9 can be written in C++ or C. When timing analysis ...

10 Session 6B: Convergence of abstractions in high-level synthesis: Application-driven processor design exploration for power-performance trade-off analysis

80%


 Diana Marculescu , Anoop Iyer

Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design November 2001

This paper presents an efficient design exploration environment for high-end core processors. The heart of the proposed design exploration framework is a two-level simulation engine that combines detailed simulation for critical portions of the code with fast profiling for the rest. Our two-level simulation methodology relies on the inherent clustered structure of application programs and is completely general and applicable to any microarchitectural power/performance simulation engine. The prop ...

11 Rapid profiling via stratified sampling

80%

 S. Subramanya Sastry , Rastislav Bodik , James E. Smith

ACM SIGARCH Computer Architecture News , Proceedings of the 28th annual international symposium on Computer architecture May 2001

Volume 29 Issue 2

Sophisticated binary translators and dynamic optimizers demand a program profiler with low overhead, high accuracy, and the ability to collect a variety of profile types. A profiling scheme that achieves these goals is proposed. Conceptually, the hardware compresses a stream of profile data by counting identical events; the compressed profile data is passed to software for analysis. Compressing the high-bandwidth event stream greatly reduces software overhead. Because optimizations can tolerate ...

12 A framework for reducing the cost of instrumented code

80%

Matthew Arnold , Barbara G. Ryder

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation May 2001
Volume 36 Issue 5

Instrumenting code to collect profiling information can cause substantial execution overhead. This overhead makes instrumentation difficult to perform at runtime, often preventing many known *offline* feedback-directed optimizations from being used in online systems. This paper presents a general framework for performing *instrumentation sampling* to reduce the overhead of previously expensive instrumentation. The framework is simple and effective, using code-duplication and *counting* ...

13 Practicing JUDO: Java under dynamic optimizations

80%

Micha? Cierniak , Guei-Yuan Lueh , James M. Stichnoth

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation May 2000
Volume 35 Issue 5

A high-performance implementation of a Java Virtual Machine (JVM) consists of efficient implementation of Just-In-Time (JIT) compilation, exception handling, synchronization mechanism, and garbage collection (GC). These components are tightly coupled to achieve high performance. In this paper, we present some static and dynamic techniques implemented in the JIT compilation and exception handling of the Microprocessor Research Lab Virtual Machine (MRL VM), ...

14 New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice

77%

Cristian Estan , George Varghese

ACM Transactions on Computer Systems (TOCS) August 2003
Volume 21 Issue 3

Accurate network traffic measurement is required for accounting, bandwidth provisioning and detecting DoS attacks. These applications see the traffic as a collection of flows they need to measure. As link speeds and the number of flows increase, keeping a counter for each flow is too expensive (using SRAM) or slow (using DRAM). The current state-of-the-art methods (Cisco's sampled NetFlow), which count periodically sampled packets are slow, inaccurate and resource-intensive. Previous work showed ...

15 Joeq: a virtual machine and compiler infrastructure

77%


John Whaley

Proceedings of the 2003 workshop on Interpreters, Virtual Machines and Emulators
June 2003

Joeq is a virtual machine and compiler infrastructure designed to facilitate research in virtual machine technologies such as Just-In-Time and Ahead-Of-Time compilation, advanced garbage collection techniques, distributed computation, sophisticated scheduling algorithms, and advanced run time techniques. Joeq is entirely implemented in Java, leading to reliability, portability, maintainability, and efficiency. It is also language-independent, so code from any supported language can be seamlessly ...

16 Enhancing the performance of 16-bit code using augmenting instructions

77%

 Arvind Krishnaswamy

ACM SIGPLAN Notices , Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems June 2003
Volume 38 Issue 7

In the embedded domain, memory usage and energy consumption are critical constraints. Dual width instruction set embedded processors such as the ARM provide a 16-bit instruction set in addition to the 32-bit instruction set to address these concerns. Using 16-bit instructions one can achieve code size reduction and I-cache energy savings at the cost of performance. We have observed that throughout 16-bit Thumb code there exist Thumb instruction pairs that are equivalent to a single ARM instructi ...

17 Continuous program optimization: A case study

77%

 Thomas Kistler , Michael Franz

ACM Transactions on Programming Languages and Systems (TOPLAS) July 2003
Volume 25 Issue 4

Much of the software in everyday operation is not making optimal use of the hardware on which it actually runs. Among the reasons for this discrepancy are hardware/software mismatches, modularization overheads introduced by software engineering considerations, and the inability of systems to adapt to users' behaviors. A solution to these problems is to delay code generation until load time. This is the earliest point at which a piece of software can be fine-tuned to the actual capabilities of the ...

18 Algorithms II: Quantifying instruction criticality for shared memory multiprocessors

77%


 Tong Li , Alvin R. Lebeck , Daniel J. Sorin

Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures June 2003

Recent research on processor microarchitecture suggests using instruction criticality as a metric to guide hardware control policies. Fields et al. [3, 4] have proposed a directed acyclic graph (DAG) model for characterizing program microexecutions on uniprocessors. Under such a model, critical path analysis can be applied and instructions' slack values can be used to quantify instruction criticality. In this paper, we extend the uniprocessor DAG model to characterize parallel program executions ...

19 Dynamic Adaptive compilation: Adaptive online context-sensitive inlining

77%

 Kim Hazelwood , David Grove

As current trends in software development move toward more complex object-oriented programming, inlining has become a vital optimization that provides substantial performance improvements to C++ and Java programs. Yet, the aggressiveness of the inlining algorithm

must be carefully monitored to effectively balance performance and code size. The state-of-the-art is to use profile information (associated with call edges) to guide inlining decisions. In the presence of virtual method calls, profile ...

20 Profile-based optimizations: Coupling on-line and off-line profile information to improve



77%

program performance

Chandra Krintz

In this paper, we describe a novel execution environment for Java programs that substantially improves execution performance by incorporating both on-line and off-line profile information to guide dynamic optimization. By using both types of profile collection techniques, we are able to exploit the strengths of each constituent approach: profile accuracy and low overhead. Such coupling also reduces the negative impact of these approaches when each is used in isolation. On-line profiling introduce ...

Results 1 - 20 of 30 short listing

 
Prev Page 1 2 Next Page

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2003 ACM, Inc.